METHOD, APPARATUS AND ARTICLES OF MANUFACTURE FOR COMPUTING THE SENSITIVITY PARTIAL DERIVATIVES OF LINKED MECHANICAL SYSTEMS

CLAIM OF PRIORITY

This application claims priority under 35 U.S.C. 119(e) to U.S. Provisional Application Serial No. 60/391,357, filed June 24, 2002, and entitled "The Application of Clifford Algebras for Computing the Sensitivity Partial Derivatives of Linked Mechanical Systems", the entire disclosure of which is hereby incorporated herein by reference.

BACKGROUND OF THE INVENTION

The calculation of sensitivity partial derivatives is a frequently occurring task during the engineering design and control process. Sensitivity models are derived from an underlying math model of the physical system. Traditionally there have been two approaches for developing the sensitivity partial derivatives: (i) analytical models, and (ii) numerical models using finite difference-like techniques. Analytical models are preferred when available, though the additional modeling, software development, and checkout are often time-consuming and expensive. Numerical methods are conceptually straightforward, however, many methods must sample a function two or more times to estimate the sensitivity.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 illustrates a plot of relative growth rates of polynomials according to one example embodiment of the invention.

FIG. 2 illustrates an example FOTRAN subroutine according to one example embodiment of the invention.

FIG. 3 illustrates an example computer platform with an Object-Oriented Cartesian Embedding Algorithm (OCEA) according to one example embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

I. INTRODUCTION

The present invention presents a third alternative approach for generating the sensitivity data: An Object-Oriented Cartesian Embedding Algorithm (OCEA). The computational advantage of the OCEA approach is that a single function evaluation yields the function value and the associated Jacobian and Hessian partial derivatives. The user is completely hidden from the details of how the partial derivatives are generated. The algorithm has been implemented in Fortran 90 and Macsyma 2.4. Module functions (Fortran 90/95) are used to encapsulate new data types, and extended math and library functions for handling vector, tensor, and embedded variables. According to certain example embodiments, the present invention supports math models for scalar, vector, linear matrix equations, and matrix inversion.

OCEA is a chain rule-based evaluation technique for analytically evaluating partial derivatives with respect to input variables of functions defined by a high-level language [Turner, 2002]. All of the problem functions are assumed to have continuous partial derivatives through second-order. Operationally, OCEA replaces conventional computer algebra operations and functions with generalized OCEA versions of these capabilities. The use of operator-overloading techniques permits a familiar conceptual framework to surround the use of OCEA software. The operator-overloading techniques redefine the intrinsic mathematical binary operators { +, -, *, **, /} and unary functions { cos(x), sin(x), tan(x), log(x),....}. New derived data types, interface operators, and generalized mathematical operators are developed for computing the first and second order partials.

Automatic differentiation has existed as a research topic since the 1980's. The common idea has been to pre-process a programmed function, identify the unary and binary functional operations, and build up forward and backward computational strategies. A successful implementation of this approach is ADIFOR (Automatic Differentiation of FORtran), which transforms the model source code based on a specification of dependent and independent variables, and produces source code that calculates the derivatives of the dependent variables with respect to the independent variables [Griewank, 89; Bischof, Carle, Corliss, Griewank, and Hovland, 92; Bischof, Carle, Khademi, Mauer, and Hovland, 95;

Eberhard and Bischof, 96; Hovland and Heath 97]. Other related codes include AD01 [Pryce and Reid, 98] which handles higher order derivatives using the chain rule and computational graphs. OCEA, unlike these previously developed tools, has the capability to build models on the fly, exploit high order models, and provide simple extensions for handling matrix calculations. No distinction is made between the independent and dependent variables. OCEA operates at the elementary scalar level, regardless of the complexity of the mathematical object.

Mathematically, OCEA replaces each scalar operation with a higher dimension object. For example, assuming the g is a 1x1 scalar in traditional computer algebra, and introducing the OCEA-based transformation process, one obtains:

$$g := \underbrace{\left[g,\right.}_{1x1} \underbrace{\vec{\nabla} g,}_{mx1} \underbrace{\left.\vec{\nabla}\vec{\nabla} g\right]}_{mxm}; \quad \vec{\nabla} = \hat{n}_1 \frac{\partial}{\partial x_1} + \cdots + \hat{n}_m \frac{\partial}{\partial x_m} \tag{1}$$

where x_i (i = 1, ...,m) denotes the vector of independent variables, $\hat{n}_i = (\delta_{i1}, \cdots, \delta_{im})$ denotes a unit vector in the i-th coordinate direction, and the transformed version of g is now of dimension $1x(1+m+m^2)$. Generalizations for higher dimensional versions of OCEA are obvious. Because of the rapid increase in the number of partial derivatives as the order of the OCEA method increases, it becomes critically important to exploit symmetry. As a point of comparison, if a q-th order version of OCEA is considered, the dimension of each transformed OCEA scalar becomes $\sum_{n=0}^{q} m^n = \left(1-m^{q+1}\right)/\left(1-m\right)$ when symmetry is not exploited. If symmetry is exploited the dimension of each transformed OCEA scalar becomes $\sum_{n=0}^{q} \binom{n+m-1}{m-1} = \binom{q+m}{m}$ where (*) denotes the binomial coefficient. A plot of the relative growth rates is presented in Figure 1.

A benefit of the OCEA operator overloading strategy is that standard programming language constructs are used in building mathematical models. The complier recognizes that new data types and automatically replaces the conventional intrinsic operators and functions with generalized OCEA intrinsic operators and functions. For example, imagine that one

wants to evaluate $f = xy/\sqrt{z}$ using standard Fortran and OCEA-enhanced Fortran, as shown in Table 1:

Math Model	$f = xy/\sqrt{z}$
Fortran f := (scalar)	$f = x*y/sqrt(z)$
OCEA-Enhanced Fortran F := (scalar, vector, tensor)	$f = x*y/sqrt(z)$ $= \left[\begin{array}{c} f, \\ \left[\begin{array}{ccc} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \end{array}\right], \\ \left[\begin{array}{ccc} \frac{\partial^2 f}{\partial x \partial x} & \frac{\partial^2 f}{\partial x \partial y} & \cdots \quad \frac{\partial^2 f}{\partial z \partial z} \end{array}\right] \end{array}\right]$

Table 1: Comparison of Fortran and OCEA Fortran Models

where three independent OCEA variables are defined (namely, x,y,z) and the Fortran and OCEA-Enhanced Fortran models are seen to be identical. The OCEA calculations, however, are very different from the Fortran calculations. The analyst never formulates or codes the partial derivatives. OCEA represents a linearized Clifford algebra.

This patent includes seven sections. Section 2 presents the mathematical foundation for OCEA. Section 3 presents the software operator overloading, data typing, interface operators, and generalized transformational operators for multiplication, division, and composite function evaluations. A differential equation sensitivity approach is presented in Section 4. A simple example is worked out in detail to highlight the operational steps required for using OCEA methodology in Section 5. Several non-trivial applications are presented in Section 6 for handling nonlinear vector functions, equation of motion generation using Lagrange's equations, and differential equations. Applications are presented in Section 7.

II. MATHEMATICAL FORMULATION

The mathematical models for the OCEA are presented in this section. An essential step in the algorithm is that the independent coordinates are transformed using Eq. (1), leading to:

$$x_i := \left[\underset{1x1}{x_i}, \underbrace{\underset{1x1}{\begin{bmatrix} \delta_{1i} \\ \vdots \\ \delta_{ni} \end{bmatrix}}}_{nx1}, \underbrace{\underset{nxn}{\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}}}_{nxn} \right]$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{1x\left(1+n+n^2\right)}$$

where δ_{ij} denotes the standard kronecker delta function and i = 1,...,n. The independent variables initialize the calculations for the embedded variables.

II.1 Intrinsic Operators and Functions

The generalizations for the intrinsic mathematical operators and functions are presented for addition, subtraction, multiplication, division, and composite functions. Addition and subtraction are straightforward. The most challenging generalizations are required for the product, division, and composite function calculations. All of the following transform equations assume that Jacobian and Hessian data for g and h has been built up from previous computational operations. One should observe that the tensor parts of the operators above frequently require vector outer products to complete the mathematical models. Many opportunities exist for exploiting the sparse structure of the resulting vector and tensor operations. The embedded coordinate mathematical operators follow as:

Addition:

$$\begin{aligned} g+h &= \left[g, \vec{\nabla}g, \vec{\nabla}\vec{\nabla}g\right] + \left[h, \vec{\nabla}h, \vec{\nabla}\vec{\nabla}h\right] \\ &= \left[g+h, \vec{\nabla}g+\vec{\nabla}h, \vec{\nabla}\vec{\nabla}g+\vec{\nabla}\vec{\nabla}h\right] \end{aligned}$$

Subtraction:

$$\begin{aligned} g-h &= \left[g, \vec{\nabla}g, \vec{\nabla}\vec{\nabla}g\right] - \left[h, \vec{\nabla}h, \vec{\nabla}\vec{\nabla}h\right] \\ &= \left[g-h, \vec{\nabla}g-\vec{\nabla}h, \vec{\nabla}\vec{\nabla}g-\vec{\nabla}\vec{\nabla}h\right] \end{aligned}$$

***Product Rule*:**

$$g*h := \left[g,\ \vec{\nabla}g,\ \vec{\nabla}\vec{\nabla}g\right] * \left[h,\ \vec{\nabla}h,\ \vec{\nabla}\vec{\nabla}h\right]$$

$$= \left[g*h,\ g*\vec{\nabla}h + h*\vec{\nabla}g,\ g*\vec{\nabla}\vec{\nabla}h + \vec{\nabla}g\left(\vec{\nabla}h\right)^T + \vec{\nabla}h\left(\vec{\nabla}g\right)^T + h*\vec{\nabla}\vec{\nabla}g\right]$$

Division Rule:

$$g/h := \left[g,\ \vec{\nabla}g,\ \vec{\nabla}\vec{\nabla}g\right] / \left[h,\ \vec{\nabla}h,\ \vec{\nabla}\vec{\nabla}h\right]$$

$$= \left[\begin{array}{c} g/h, \\ \left(\dfrac{h\vec{\nabla}g - g\vec{\nabla}h}{h^2}\right), \\ \left(\dfrac{h\vec{\nabla}\vec{\nabla}g - g\vec{\nabla}\vec{\nabla}h - \vec{\nabla}g\left(\vec{\nabla}h\right)^T - \vec{\nabla}h\left(\vec{\nabla}g\right)^T}{h^2}\right) + \left(\dfrac{2*g\vec{\nabla}h\left(\vec{\nabla}h\right)^T}{h^3}\right) \end{array}\right]$$

Composite Function Rule:

$$f\left(\left[g,\ \vec{\nabla}g,\ \vec{\nabla}\vec{\nabla}g\right]\right) := \left[f(g),\ \frac{\partial f}{\partial g}\vec{\nabla}g,\ \frac{\partial^2 f}{\partial g^2}\vec{\nabla}g\left(\vec{\nabla}g\right)^T + \frac{\partial f}{\partial g}\vec{\nabla}\vec{\nabla}g\right]$$

where the three scalar values for f(g), $\partial f/\partial g$, and $\partial^2 f/\partial g^2$ are computed using standard intrinsic mathematical functions.

II.2 Example Embedded Function Calculations

This section considers the calculation of embedded versions of $\sqrt{g}$ and $\tan^{-1}(g)$. The square root calculation is considered first.

II.2.1 $\sqrt{g}$ *Calculation:* Referring to the composite function rule above one has $f(g) = \sqrt{g}$, which leads to $\frac{\partial f}{\partial g} = \frac{1}{\sqrt{g}}$, and $\frac{\partial^2 f}{\partial g^2} = \frac{-1}{2 * g^{3/2}}$, so that the embedded composite square root function is given by:

$$\sqrt{g} := \sqrt{\left[g, \vec{\nabla} g, \vec{\nabla}\vec{\nabla} g\right]} = \left[\sqrt{g}, \frac{\vec{\nabla} g}{\sqrt{g}}, \frac{-\vec{\nabla} g \left(\vec{\nabla} g\right)^T}{2 * g^{3/2}} + \frac{\vec{\nabla}\vec{\nabla} g}{\sqrt{g}}\right]$$

The $\tan^{-1}(g)$ calculation is considered next.

II.2.2 $\tan^{-1}(g)$ *Calculation:* Referring to the composite function rule above one has $f(g) = \tan^{-1}(g)$, which leads to $\frac{\partial f}{\partial g} = \frac{1}{1+g^2}$, and $\frac{\partial^2 f}{\partial g^2} = \frac{-2g}{(1+g^2)^2}$, so that the embedded composite $\tan^{-1}(g)$ function is given by:

$$\tan^{-1}(g) := \tan^{-1}\left(\left[g, \vec{\nabla} g, \vec{\nabla}\vec{\nabla} g\right]\right) = \left[\tan^{-1}(g), \frac{\vec{\nabla} g}{1+g^2}, \frac{-2g\vec{\nabla} g \left(\vec{\nabla} g\right)^T}{\left(1+g^2\right)^2} + \frac{\vec{\nabla}\vec{\nabla} g}{1+g^2}\right]$$

Similar generalizations have been developed for the operators "**", sin, cos, tan, asin, acos, exp, log, sinh, cosh, and tanh. Others can be easily developed as required.

III. SOFTWARE ARCHITECTURE

According to one example embodiment, the OCEA mathematical operators and functions are translated into an object-oriented language where operator-overloading techniques are employed to re-define the algebraic operations that control the computational procedures. According to one embodiment, A Fortran 90 version of OCEA may be provided. An OCEA program is shown executing on a suitable computing platform, such as a personal computer or workstation, or other computing system, in Figure 3, wherein the software is stored in a machine readable form in memory, on a magnetic or optical disk, or in the form of a digital signal in electrical form. Defined data types, interface operator definitions, and module functions are used for encapsulating the data types and providing a language extension that supports the automatic generation of partial derivatives. As shown in Section II, according to one example embodiment, three derived data types are required: vector, tensor, and embedded. The embedded capabilities provide the OCEA capabilities for

libraries of mathematical operators and functions. The introduction of derived data types provides a software advantage; namely, that the compiler can detect the data types involved in a calculation and invoke the correct subroutine or function without user intervention. This is achieved in the software architecture design by providing typed data subroutines and functions for covering all possible mathematical operations. (i.e., real and embedded, embedded and embedded, and so on). For example, Table 2 provides a partial list of the capabilities required for generalizing the assignment (=) and math operators. An additional advantage is that the compiler can use array-processing techniques for handling large problems.

Data Types (A&B)	Operations	Math Model
Embedded, Embedded	"+","-","="	A+B, A-B
Embedded, Vector	"="	A=B
Vector, Embedded	"="	A=B
Vector, Scalar	"*","="	A=b*A, A_i=b, i=1,…,n
Tensor, Tensor	"="	A=B
Embedded, Embedded	"sin"	A=sin(B)

Table 2: Operator Overloading for Data Type Operations

Two detailed examples are provided for clarifying the procedure. The first example presents vector handling for adding two vectors and the second supports multiplying a tensor by a scalar. **PV** denotes the defined vector data type, where **VPART** is the vector structure constructor variable. **PT** denotes the defined tensor data type, where **TPART** is the tensor structure constructor variable. The variable ":" denotes an array assignment where all components are assigned. Mixed data types are not allowed. Explicit data typing is required for all variables. As a preprocessing step, the compiler finds the correct function by (i) identifying the operator, and (ii) checking for module procedures with the same the input data types.

III.1 Adding Vector Data Types

If the compiler detects that two vector data types are added and that the result is a vector data type, then an automatic link is generated those calls the function routine FUNCTION ADD_PV in Module PV_HANDLING. Symbolically this equation looks like

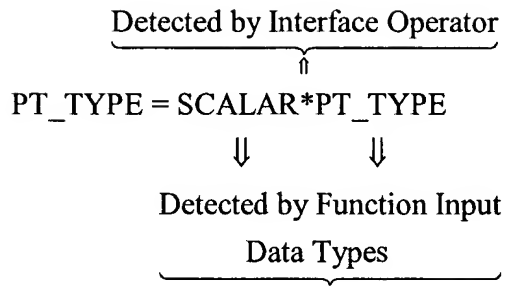
$$\underbrace{\text{PV\_TYPE} = \text{PV\_TYPE} + \text{PV\_TYPE}}_{\Uparrow \text{ Detected by Interface Operator}}$$
$$\Downarrow \qquad \Downarrow$$
$$\underbrace{\text{Detected by Function Input Data Types}}$$

When vector data types are added, an interface operator for addition is defined and a name is assigned for the module procedure that takes two vector data types as input, as presented in the software fragment:

```
MODULE PV_HANDLING   !Vector Operations
    .
    .
    INTEFACE OPERATOR (+)
        MODULE PROCEDURE ADD_PV
    END INTERFACE
    .
    .
    FUNCTION ADD_PV(A,B)
        TYPE(PV)  ADD_PV
        TYPE(PV), INTENT(IN)::A,B
        ADD_PV%VPART(:) = A%VPART(:) + B%VPART(:)
    END FUNCTION ADD_PV
    .
    .
END MODULE PV_HANDLING
```

III.2 Multiplying a Tensor Data Type by a Real Scalar

If the compiler detects that a real scalar multiplies a tensor data type and that the result is a tensor data type, an automatic link is generated that calls the function routine FUNCTION_R_PT in Module PT_HANDLING. Symbolically this equation looks like



When a real variable and a tensor are multiplied, an interface operator is defined for multiplication and a name is assigned for the module procedure available for multiplying a tensor by a scalar, as presented in the software fragment:

```
MODULE PT_HANDLING    !Tensor Operations
    .
    .
    INTERFACE OPERATOR (*)
       MODULE PROCEDURE MULT_R_PT
    END INTERFACE
    .
    .
    FUNCTION MULT_R_PT(R,A)
    TYPE(PT)         MULT_R_PT
    REAL(KIND=8), INTENT(IN)::R
    TYPE(PT),        INTENT(IN)::A
    INTEGER::I
    DO I = 1, N
       MULT_R_PT%TPART(:,I) = R*A%TPART(:,I)
    END DO
END FUNCTION MULT_R_PT
.
.
END MODULE PT_HANDLING
```

The object oriented data typing hides the building of links to the specific subroutines and functions required for completing calculations.

IV. DIFFERENTIAL EQUATION SENSITIVITY

Linked mechanical systems lead to first-order differential equations of the form:

$$\dot{x} = M(x)^{-1} f(x,t) \tag{2}$$

Partial derivatives of Eq. (2) are required for computing state transition matrices and high-dimensional sensitivity operators. Even at second-order the partial derivatives can become quite involved. Using OCEA versions of x, f, and M, one obtains

$$\dot{x} := \left[M^{-1} f, \quad \bar{\nabla}\left(M^{-1} f\right), \quad \bar{\nabla}\bar{\nabla}\left(M^{-1} f\right) \right]$$

The value of the implicit solution is that the inverse of M(x) is never formed explicitly or manipulated to generate the partial derivatives. The complexity of the implicit solution is compared with a conventional sensitivity approach.

IV.1 OCEA Gaussian Elimination Approach:

The calculation for Eq. (2) is simplified by redefining the equation as

$$M(x)\dot{x} = f(x,t) \tag{3}$$

where OCEA versions of the variables are defined as

$$\dot{x} := \left[\dot{x}, \left[\dot{x}_{,x_1} \quad \cdots \quad \dot{x}_{,x_n} \right], \left[\dot{x}_{,x_1,x_1} \quad \dot{x}_{,x_1,x_2} \quad \cdots \quad \dot{x}_{,x_n,xn_1} \right] \right]$$

$$f := \left[f, \left[f_{,x_1} \quad \cdots \quad f_{,x_n} \right], \left[f_{,x_1,x_1} \quad f_{,x_1,x_2} \quad \cdots \quad f_{,x_n,xn_1} \right] \right]$$

and

$$M := \left[M, \left[M_{,x_1} \quad \cdots \quad M_{,x_n} \right], \left[M_{,x_1,x_1} \quad M_{,x_1,x_2} \quad \cdots \quad M_{,x_n,xn_1} \right] \right]$$

Equation (3) is solved by using an OCEA version of Gaussian elimination. The Gaussian elimination algorithm is identical to standard algorithms, except that a single Use EB_Handling Module command is introduced and variables are typed as needed. The generalized intrinsic operators handle all of the details. The resulting calculation builds the partial derivatives of the state differential equation, without having to form M^{-1} or any matrix products. A numerical example is given in Section VI.

IV.2 Conventional Second Order Calculation:

Analytically computing the second-order partial derivatives of Eq. (3) leads to

$$\dot{x}_{,p,q} = \left(M^{-1}\right)_{,p,q} f + \left(M^{-1}\right)_{,p} f_{,q} + \left(M^{-1}\right)_{,q} f_{,p} + M^{-1} f_{,p,q} \tag{4}$$

where

$$\left(M^{-1}\right)_{,p} = -M^{-1} M_{,p} M^{-1}$$

and

$$\left(M^{-1}\right)_{,p,q} = M^{-1} M_{,p} M^{-1} M_{,q} M^{-1} + M^{-1} M_{,q} M^{-1} M_{,p} M^{-1} - M^{-1} M_{,p,q} M^{-1}$$

By counting the operations for the second-order inverse matrix partials, one finds that $(5n^2 + 11n)/2$ nxn matrix products must be formed (fully accounting for symmetry). Consequently, for a medium-sized system where n = 100, one must evaluate 25,550 100x100 matrix products. Comparing Eqs. (2) and (4), one observes that the OCEA solution generates the right-hand-side of Eq. (4) without ever explicitly forming any of the complicated matrix products presented above.

IV.3 State and Parameter Transition Matrix Calculations

The state and parameter transition matrix calculations are presented in this section. Both analytical and OCEA models are presented for generating the partial derivatives. Because the transition matrix variables are implicitly defined, OCEA is used to generate the information required for building the math models for integrating the partials. For simplicity, only a first order version of OCEA is assumed for the derivations presented in Sections IV.3.1 through IV.3.3.

IV.3.1 Analytic State Transition Matrix: The state transition matrix is obtained by integrating a first order model of the form

$$\dot{x} = F(x,t), \; x_o = x\big|_{t=t_0} \tag{5}$$

where x is the nx1 state vector and the sensitivity of the terminal state with respect to the initial state is required. Analytically the required partial derivatives are obtained by integrating Eq. (5) as:

$$x(t) = x(t_0) + \int_{t_0}^{t} F(x(\tau), \tau) d\tau$$

and computing the partial derivative with respect to $x(t_0)$, leading to:

$$\frac{\partial x(t)}{\partial x(t_0)} = I + \int_{t_0}^{t} \frac{\partial F}{\partial x(\tau)} \frac{\partial x(\tau)}{\partial x(t_0)} d\tau$$

where the initial condition is defined by an nxn identity matrix. The governing differential equation for the state transition matrix is obtained by differentiating the equation above with respect to t, yielding the nxn matrix differential equation:

$$\frac{d}{dt}\left(\frac{\partial x(t)}{\partial x(t_0)}\right) = \frac{\partial F}{\partial x(t)} \frac{\partial x(t)}{\partial x(t_0)}; \quad \frac{\partial x(t)}{\partial x(t_0)}\Big|_{t_0} = I \tag{6}$$

IV.3.2 Analytic Parameter State Transition Matrix: The parameter transition matrix is obtained by integrating a first order model of the form

$$\dot{x} = F(x, t; p), \; x_o = x|_{t=t_0}, \; p \text{ is a } (mx1) \text{ vector of parameters} \tag{7}$$

where x is the nx1 state vector and the sensitivity of the terminal state with respect to the problem parameters is required. Analytically the required partial derivatives are obtained by integrating Eq. (7) as:

$$x(t) = x(t_0) + \int_{t_0}^{t} F(x(\tau), \tau; p) d\tau$$

and computing the partial derivative with respect to p, leading to:

$$\frac{\partial x}{\partial p} = \int_{t_0}^{t} \left(\frac{\partial F}{\partial x(\tau)} \frac{\partial x(\tau)}{\partial p} + \frac{\partial F}{\partial p} \right) d\tau$$

where the initial condition is defined by an nxm zero matrix. The governing differential equation for the parameter transition matrix is obtained by differentiating the equation above with respect to t, yielding the nxm matrix differential equation:

$$\frac{d}{dt}\left(\frac{\partial x}{\partial p}\right)=\frac{\partial F}{\partial x(\tau)}\frac{\partial x(\tau)}{\partial p}+\frac{\partial F}{\partial p};\quad \frac{\partial x}{\partial p}\Big|_{t_0}=0 \tag{8}$$

IV.3.3 OCEA State and Parameter Transition Matrix Calculations: The state transition matrix differential equation of Eq. (6) is obtained from an OCEA version of Eq. (5), as follows:

$$\begin{aligned}\dot{x}&=F\\ \dot{x}&:=\left[\dot{x}\quad \vec{\nabla}\dot{x}\ \right]=\left[\dot{x}_1\quad \dot{x}_2\right]\\ F&:=\left[F\quad \vec{\nabla}F\right]=\left[F_1\quad F_2\right]\end{aligned}$$

where the subscripts denote the different parts of the OCEA variable. By defining the partial derivative variable $\partial x/\partial x_0$, the required differential equations are given by

$$\begin{aligned}\dot{x}_1&=F_1 \quad ;\quad x_1\big|_{t_0}=x_0\\ \frac{d}{dt}\frac{\partial x}{\partial x_0}&=F_2\frac{\partial x}{\partial x_0}\ ;\ \frac{\partial x}{\partial x_0}\Big|_{t_0}=I_{nxn}\end{aligned} \tag{9}$$

The parameter transition matrix differential equation of Eq. (8) is obtained from an OCEA version of Eq.(7), as follows:

$$\begin{aligned}\dot{x}&=F\\ \dot{x}&:=\left[\dot{x}\quad \vec{\nabla}\dot{x}\ \right]=\left[\dot{x}\quad \vec{\nabla}_x\dot{x}\quad \vec{\nabla}_p\dot{x}\ \right]=\left[\dot{x}_1\quad \dot{x}_2\quad \dot{x}_3\right]\\ F&:=\left[F\quad \vec{\nabla}F\right]=\left[F\quad \vec{\nabla}_xF\quad \vec{\nabla}_pF\ \right]=\left[F_1\quad F_2\quad F_3\right]\end{aligned}$$

where OCEA variables now include both x and p--which allows the gradient operator to be split into state and parameter parts, where the subscripts denote the different parts of the OCEA variable. By defining the partial derivative variable $\partial x/\partial p$, the required differential equations are given by

$$\begin{aligned}\dot{x}_1&=F_1 \quad ;\quad x_1\big|_{t_0}=x_0\\ \frac{d}{dt}\frac{\partial x}{\partial p}&=F_2\frac{\partial x}{\partial p}+F_3\ ;\ \frac{\partial x}{\partial p}\Big|_{t_0}=0_{nxn}\end{aligned} \tag{10}$$

For both Eqs. (9) and (10) OCEA builds the complex and tedious part of the calculation automatically.

V. SIMPLE EXAMPLE PROBLEM:

To highlight the required embedded calculations, the following simplified 3D example application is presented. The goal is to compute an embedded version of the composite function $y\sqrt{xz}$, where the embedded versions of x, y, and z follow as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} := \begin{pmatrix} \left[x, [1,0,0], \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \\ \left[y, [0,1,0], \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \\ \left[z, [0,0,1], \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] \end{pmatrix}$$

The following three steps are required:

Step 1: Form the product x*z (using the product operator):

$$x*z := \left[x, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right] * \left[z, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right]$$

$$= \begin{bmatrix} x*z,\ x* \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + z* \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \\ x* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^T + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}^T + z* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{bmatrix}$$

$$= \left[x*z, \begin{bmatrix} z \\ 0 \\ x \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \right]$$

Step 2: Compute the $\sqrt{x*z}$ (using the composite function operator):

$$\sqrt{x*z} := \left[\sqrt{x*z}, \frac{1}{2*\sqrt{x*z}}\begin{bmatrix} z \\ 0 \\ x \end{bmatrix}, \frac{-1}{4*(x*z)^{3/2}}\begin{bmatrix} z \\ 0 \\ x \end{bmatrix} * \begin{bmatrix} z \\ 0 \\ x \end{bmatrix}^T + \frac{1}{2*\sqrt{x*z}}\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}\right]$$

$$= \left[\sqrt{x*z}, \begin{bmatrix} \frac{z}{2*\sqrt{x*z}} \\ 0 \\ \frac{x}{2*\sqrt{x*z}} \end{bmatrix}, \begin{bmatrix} \frac{-z^2}{4*(x*z)^{3/2}} & 0 & \frac{1}{2*\sqrt{x*z}} - \frac{x*z}{4*(x*z)^{3/2}} \\ 0 & 0 & 0 \\ \frac{1}{2*\sqrt{x*z}} - \frac{x*z}{4*(x*z)^{3/2}} & 0 & \frac{-x^2}{4*(x*z)^{3/2}} \end{bmatrix}\right]$$

Step 3: Compute the product $y*\sqrt{x*z}$ (using the product operator):

$$y*\sqrt{x*z} := \left[y*\sqrt{x*z}, \begin{bmatrix} \frac{y*z}{2*\sqrt{x*z}} \\ \sqrt{x*z} \\ \frac{x*y}{2*\sqrt{x*z}} \end{bmatrix}, \begin{bmatrix} \frac{-y*z}{4*x*\sqrt{x*z}} & \frac{z}{2*\sqrt{x*z}} & \frac{y*\sqrt{x*z}}{4*x*z} \\ \frac{z}{2*\sqrt{x*z}} & 0 & \frac{x}{2*\sqrt{x*z}} \\ \frac{y*\sqrt{x*z}}{4*x*z} & \frac{x}{2*\sqrt{x*z}} & \frac{-x*y}{4*z*\sqrt{x*z}} \end{bmatrix}\right]$$

The calculations can become extremely complex; nevertheless, the user is completely hidden from the details of the calculations and the calculations are accurate to the working precision of the machine.

VI. COMPLEX APPLICATIONS

Three classes of applications have been tested using a Fortran 90 version of OCEA. The three application classes consist of: (1) Non-linear n-dimensional vector Jacobian and Hessian calculations. (2) Generation of Lagrange's equations of motion, and (3) Generation of state transition matrix partials. These examples are presented in Section VI.1 through VI.3.

VI.1 Nonlinear Function Evaluation

A highly nonlinear 7x1 vector algebraic function, $F(\xi)$, of five independent variables, ξ, is considered where $F(\xi)$ is given by:

$$F(\xi)=\begin{pmatrix} e^{u}+x^{2}*\cos(z) \\ x*u*(y*z)^{2} \\ x*y*z^{2} \\ z^{3}/u \\ z*\sqrt{w} \\ x*u*\sin^{-1}\left(y/(u*w)\right) \\ z^{1/3}*\ln\left(\sqrt{u}\right) \end{pmatrix}$$

and $\xi=(x,y,z,u,w)$. The computational procedure is outlined in Figure 2, which consists of a Fortran 90 subroutine for executing the OCEA calculations for the function defined above. The user inputs a real-valued vector of independent variable (i.e., INDVAR). These variables are internally transformed to OCEA variables for the calculation (i.e., EB_VAR). NV denotes the number of independent variables and NF denotes the number of function dimensions to be evaluated, where both are defined in Module Problem_Data. The Module EB_Handling provides all of the OCEA operator overloading and extended function capabilities. The outputs for the routine are: FX-the vector function (7x1), JAC-the Jacobian (7x5), and HES-the Hessian (5x5x7). Numerical results from the subroutine below have been compared with symbolically generated models for the same system using the Macsyma computer algebra system; no errors have been found. For this class of problems the user inputs data in the normal way, writes a math model for evaluating the function, and receives

the function, Jacobian, and Hessian without knowing any details about how the calculation is carried out.

VI.2 Lagrange's Equations of Motion

Lagrange's methods is a classical analytical dynamics method that requires the use of first and second order partial derivatives for the building the equations of motion for linked subsystems. A simplified analysis is presented because no rotating frame derivatives are considered. If m-generalized coordinates are available for analyzing the motions of the physical system, the velocity coordinates follow as:

$$\underline{v}_i = \underline{v}_i(q, \dot{q}, t) \; ; \; i = 1, \cdots, n$$

where q denotes the mx1 vector of generalized coordinates and $\dot{q}$ denotes the time derivative of q. The velocity vector is used in building the kinetic energy, leading to:

$$T = T(q, \dot{q})$$

The calculations for Lagrange's equations are simplified by defining q and $\dot{q}$ as OCEA variables, and using these variables to evaluate the velocity and kinetic energy. The OCEA version of the kinetic energy becomes:

$$T := \left[T, \; \begin{bmatrix} T_{,q} \\ T_{,\dot{q}} \end{bmatrix}, \; \begin{bmatrix} T_{,q,q} & T_{,q,\dot{q}} \\ T_{,\dot{q},q} & T_{,\dot{q},\dot{q}} \end{bmatrix} \right] \tag{11}$$

which is immediately useful for evaluating Lagrange's in the form:

$$\left(T_{,\dot{q},\dot{q}}\right) \cdot \ddot{q} = Q + T_{,q} - \left(T_{,q,\dot{q}}\right) \cdot \dot{q}$$

where the generalized force is given by $Q_q = \sum_{i=1}^{N_b} F_i^T v_{i,q}$ and the velocity partial derivative is obtained from the OCEA version of the velocity. This OCEA-based application produces an enormous reduction in the normal labor required for generating Lagrange's equations.

Similar labor reductions appear for Hamilton's method. A 2D example is presented in the next section.

VI.2.1 2D Example

Assume that a cart of mass m_1 rolls without friction on a smooth surface. A linear spring restricts the cart motion in the x-axis direction. A massless rod of length l is attached to the cart and has a tip mass m_2. The rod freely rotates about the attachment point to the cart. Gravity is the only assumed external force. The OCEA variables for the problem are $\left(x,\theta,\dot{x},\dot{\theta}\right)$. The particle position vectors are given by:

$$r_1 = (x, 0)$$
$$r_2 = (x + l * \cos(\theta), -l * \sin(\theta))$$

and the velocity variables are given by

$$v_1 = (\dot{x}, 0)$$
$$v_2 = (\dot{x} - l * \sin(\theta) * \dot{\theta}, -l * \cos(\theta) * \dot{\theta})$$

T o compute the kinetic energy the velocity variables are transformed to OCEA form, where the individual velocity components are given by

$$v_{1x} := \left[\dot{x},\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right]$$

$$v_{1y} := \left[0,\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right]$$

$$v_{2x} := \left[\dot{x} - l*\sin(\theta)*\dot{\theta},\ \begin{bmatrix} 0 \\ -l*\cos(\theta)*\dot{\theta} \\ 1 \\ -l*\sin(\theta) \end{bmatrix},\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & l*\sin(\theta)*\dot{\theta} & 0 & -l*\cos(\theta) \\ 0 & 0 & 0 & 0 \\ 0 & -l*\cos(\theta) & 0 & 0 \end{bmatrix} \right]$$

$$v_{2y} := \left[-l*\cos(\theta)*\dot{\theta},\ \begin{bmatrix} 0 \\ l*\sin(\theta)*\dot{\theta} \\ 0 \\ -l*\cos(\theta) \end{bmatrix},\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & l*\cos(\theta)*\dot{\theta} & 0 & l*\sin(\theta) \\ 0 & 0 & 0 & 0 \\ 0 & l*\sin(\theta) & 0 & 0 \end{bmatrix} \right]$$

The kinetic energy for this simple system is given by

$$T = m_1(v_{1x}v_{1x} + v_{1y}v_{1y})/2 + m_2(v_{2x}v_{2x} + v_{2y}v_{2y})/2$$

Using OCEA algebra leads to

$$T := \left[\left(m_1 + m_2\right)\dot{x}^2/2 - lm_2\sin(\theta)\dot{x}\dot{\theta} + l^2 m_2\dot{\theta}^2/2,\ \begin{bmatrix} 0 \\ -lm_2\cos(\theta)\dot{x}\dot{\theta} \\ \left(m_2 + m_1\right)\dot{x} - lm_2\sin(\theta)\dot{\theta} \\ l^2 m_2\dot{\theta} - lm_2\sin(\theta)\dot{x} \end{bmatrix},\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & lm_2\sin(\theta)\dot{x}\dot{\theta} & -lm_2\cos(\theta)\dot{x}\dot{\theta} & -lm_2\cos(\theta)\dot{x} \\ 0 & -lm_2\cos(\theta)\dot{\theta} & m_2 + m_1 & -lm_2\sin(\theta) \\ 0 & -lm_2\cos(\theta)\dot{x} & -lm_2\sin(\theta) & l^2 m_2 \end{bmatrix} \right]$$

where the vector and matrix partitions are easily obtained. In a real engineering application, the elements of T are numbers that are immediately used for evaluating the dynamic response.

VI.3 Equation of Motion Sensitivity.

Given the first-order differential equation of the form

$$M(x)\dot{x} = f$$

where the 3x3 matrix M(x) is given by

$$M(x) = \begin{bmatrix} 100x_1 & 5\sin(x_1x_2) & 50x_2x_3 \\ & 50x_2^2 & x_3^2 \\ SYM. & & x_1x_2x_3 \end{bmatrix}$$

and the 3x1 vector f(x) is given by

$$f(x) = \begin{pmatrix} \exp(-x_2/2)\sin(x_3) \\ -10\sqrt{x_1/x_3} \\ -10\cos(x_3^2) \end{pmatrix}$$

These equations are evaluated using OCEA algebra and the linear matrix equation is solved by Gaussian elimination. The results have been compared with the Macsyma computer algebra system and all first and second order partial derivatives have been correctly evaluated. Because of the limitations of space, only representative results are presented. Assume that $(x_1,x_2,x_3) = (1,2,3)$ as numerical values. The rate vector becomes

$$\dot{x} = \begin{pmatrix} -0.34422E-02 \\ -0.28868E-01 \\ 0.17580E-02 \end{pmatrix}$$

The partial derivative of the rate vector with respect to x_2 follows as

$$\frac{\partial \dot{x}}{\partial x_2} = \begin{pmatrix} 0.87449E-03 \\ 0.28898E-01 \\ -0.189552E-02 \end{pmatrix}$$

and the partial derivative of the rate vector with respect to x_2 and x_3 follows as

$$\frac{\partial^2 \dot{x}}{\partial x_2 \partial x_3} = \begin{pmatrix} -0.36666E-02 \\ -0.47731E-02 \\ 0.43682E-02 \end{pmatrix}$$

These results are typical. All of the first and second order partials have been validated by analytically inverting M(x), multiplying the result by f(x), and explicitly evaluating the partial derivative.

VII Applications

OCEA provides a rational process for generating sensitivity models by creating a new level of scientific and engineering software data abstraction and language extension. It has broad potential use for impacting the design and use of mathematical programming tools for applications in science and engineering. OCEA software replaces a time-consuming, error-prone, labor-intensive, and costly endeavor, with an automated tool that generates exact arbitrarily complex partial derivatives models. By developing the intrinsic operations at the scalar elemental level, the algorithm easily extends to Matrix applications. Future generalization will consider large-scale sparse applications.

Bibliography (this list of publications are not admitted to be prior art):

1. James Turner, "Object Oriented Coordinate Embedding Algorithm For Automatically Generating the Jacobian and Hessian Partial Derivatives of Nonlinear Vector Function," Invention Disclosure, University of Iowa, May 2002.
2. Andreas Griewank. "On Automatic Differentiation" in Mathematical Programming: Recent Developments and Applications, edited by M. Iri and K. Tanabe, pgs. 83-108, Kluwer Academic Publishers, Amsterdam, 1989.
3. Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland, "ADIFOR: Generating Derivative Codes from Fortan Programs," Scientific Programming, 1:11-29, 1992.
4. Cristian Bishchof, Alan Carle, Peyvand Khademi, Andrew Mauer, and Paul Hovland. "ADIFOR 2.0 User's Guide (Revision CO, " Technical Report ANL/MCS-TM-192, Mathematics and computer Science Division, Argonne National Laboratory, Argonne, IL., 1995.
5. Peter Eberhard and Christian Bischof. "Automatic Differentiation of Numerical Integration Algorithms," Technical Report ANL/MCS-P621-1196, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1996.
6. Paul Hovland and Michael Heath."Adaptive SOR: A Case Study in Automatic Differentiation of Algorithm Parameters, "Technical report ANL/MCS-P673-0797, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 1997.
7. J.D. Pryce and J.K. Reid. AD01, a Fortran 90 code for automatic differentiation. Report RAL-TR-1998-057, Rutherford Appleton Laboratory, UK, 1998.